Collaboration Strategies for Distributed Teams
A Case Study of CAD Systems Integration

Kurt E. Madsen
www.metatech.us
Tampa, Florida USA
kmadsen@metatech.us

Abstract

Software development becomes more challenging when teams are geographically distributed. One cause is that certain activities in the development lifecycle are inherently collaborative. This paper presents our experiences applying strategies that improved collaboration on our global team. Our experiences are drawn from a project in which we developed a system to integrate a computer-aided design tool with an online parts library.

First, we identify common barriers to collaboration and the challenges they presented to our team. Barriers such as language differences, time zones, and miscommunication hindered our progress by creating challenges such as ambiguity in requirements, misunderstandings of design intent, and rework. Next, we present the strategies we used to overcome these barriers. Our approach led to a successful project with a few detours along the way.

Keywords

Offshore product development, distance learning, remote collaboration, project management, security, decision tables, and Computer-Aided Design (CAD).

1. Introduction

Mechanical engineers use CAD tools to design 2D and 3D computer models of products. Examples of CAD tools include Pro/Engineer, SolidWorks, and Blender. Once a product has been designed, the model that describes it is sent to a factory for manufacturing. An interesting aspect of CAD models is that they are recursive: a model can contain a single part, an assembly of parts, an assembly of assemblies of parts, and so on. For example, an automobile is an assembly with many parts such as doors, wheels, and an engine. The engine, in turn, is also assembly that has many parts such as pistons, values, and a crankshaft.

Mechanical engineers collaborate by sharing and reusing parts to create new CAD models. Further, a single model is often edited and reviewed by many mechanical engineers before it is approved and sent to the factory for manufacturing. This sharing and aggregation of parts into assemblies introduces versioning problems similar to software configuration management. The system described in this paper provides a managed environment in which engineers can check CAD models into and out of an online parts library.

1.1. System overview

Figure 1 shows the deployment view of our CAD integration software. Our users, the mechanical engineers, view the user interfaces for the CAD tool and on-line parts library side-by-side. They reuse parts in the library to build assemblies in the CAD tool.

The server for the parts library consists of a web application and a database to store the CAD files with associated meta-data. Meta-data describes the CAD files (e.g., file version, owner, model parameters).

The desktop software includes a supported CAD tool and a browser such as Internet Explorer or Firefox. The browser provides access to the on-line parts library. Our integration software runs as a Java applet launched by the browser. This applet has two parts: the adapter framework interacts with JavaScript downloaded from the server; the CAD plug-in communicates with the CAD tool.

978-0-7695-3551-7/09 $25.00 © 2009 IEEE
DOI 10.1109/ICONS.2009.46

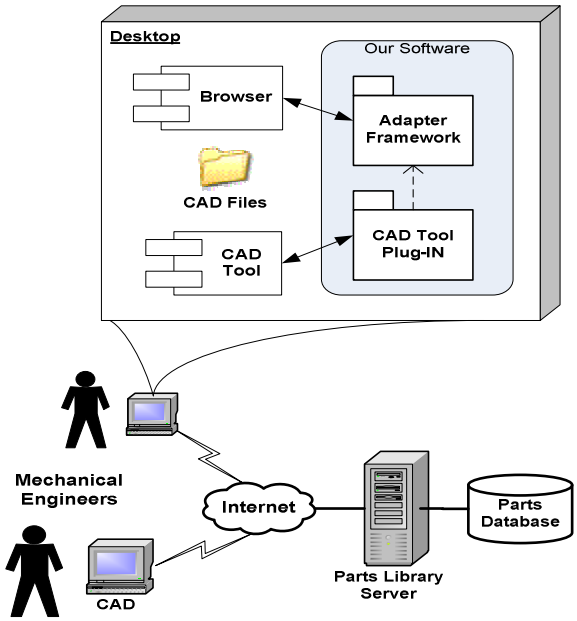

Figure 1. Deployment View

1.2. Barriers to remote collaboration

It is more difficult to work with people far away than close by. This is one reason that companies are more likely to send coding and testing activities offshore than requirements gathering and design [1]. Activities early in the software development lifecycle involve higher degrees of collaboration.

In our case, we engaged offshore resources early in the project lifecycle because they had detailed knowledge of CAD tools, which we lacked. We needed to understand what was possible before we could design a solution to fit our needs.

Given this, our team struggled through several barriers to remote collaboration including time zones, cultural barriers, miscommunication due to language gaps, mismatched skill sets, and schemata differences. (Schemata are personal, experience-based, conceptual frameworks for problem solving [2].) These barriers led to project challenges in the following areas: ambiguity in requirements, misunderstandings of design intent, quality issues, and some rework.

We used these strategies to improve collaboration:

- ***Eliminate ambiguity*** – Use unambiguous artifacts such as code-generated documents and decision tables to build common understanding even when requirements are evolving.
- ***Design competition*** – Each remote site competes to design different solutions to the same problem. Then, the resulting research and designs are compared and merged. This strategy improves quality, saves time, and builds consensus (since dissenting team members must either produce or be quiet).
- ***Strategy map*** – This strategy tool maps the cause-effect chain of a team's collaboration activities to other activities in four company perspectives: learning, internal operations, customer focus, and financial. The goal is to minimize barriers to collaboration.

The rest of this paper explains how we applied these strategies to improve collaboration across our geographically distributed team. For each strategy, we describe a sample problem along with our approach to solving it. Finally, we map the results back to our collaboration barriers and challenges.

2. Eliminate ambiguity

Some cultures encourage people to look at things in unique and different ways – to tolerate ambiguity [4]. This is both a strength and a weakness. It is a cultural barrier to collaboration because ambiguity makes collaboration on distributed teams difficult.

Sources of ambiguity include evolving requirements and design ideas. Around 500 B.C., Heraclitus wrote that "Everything flows and nothing remains the same" [5]. Kent Beck describes flow in software development as a continuous flow of [extreme programming] activities rather than discrete lifecycle phases [6].

Yet, most software developers attempt to freeze requirements in the analysis phase, largely because it takes a long time to translate requirements into a working system. By the time a system is delivered, the requirements have often changed. Frequent changes plus miscommunication are some of the reasons that the anticipated benefits of software outsourcing such as reduced cost often fail to materialize. [7]. Thus, our first collaboration strategy was to eliminate ambiguity, yet tolerate changing requirements.

2.1. Problem description

Figure 2 shows the protocol between the CAD tool adapter framework and the online parts library.

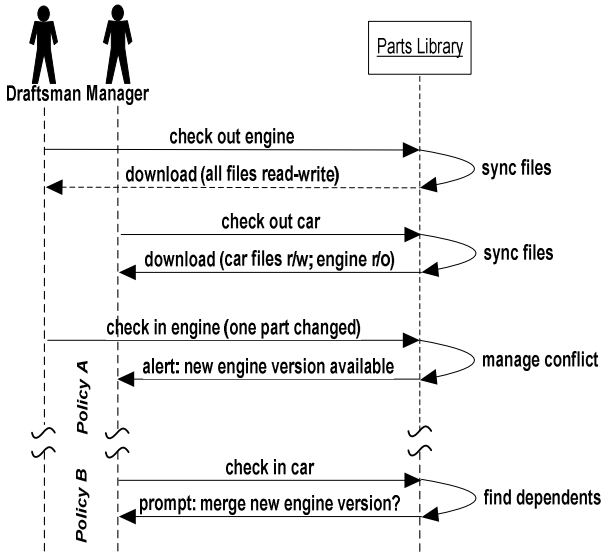

Figure 2. Protocol to check parts into the library

In this scenario, two users work together on a CAD model of a car. First the draftsman checks out the engine and gets all CAD files in read-write mode. Then, the manager attempts to check out the car, but the engine files are returned in read-only mode (since they are currently checked out by the draftsman). Next, the draftsman checks in the new version of the engine model after changing one part.

At this point, the system can execute one of two alternatives determined by policy settings. It could immediately alert the manager that the engine model has changed and offer the option to check out the new engine version. Alternatively, the system could wait until the manager later attempts to check in the car. Then it would prompt the manager to select a version of the engine to which the car should be bound.

2.2. Our approach

For each use case, we documented the client/server protocol's finite state machine, web services, XSD schema, and XML sample data. One programmer coded the client interface in Java; another coded the parts library interface in .NET/VB. But, their ability to communicate was limited by language, time zones, and cultural barriers. Despite good documentation and conference calls, the prototype code had integration issues and poor implementation. Client/server logic was hard-coded based on state variables that were passed back and forth as hidden text fields over HTTP. Debugging and testing the system was difficult.

To reduce ambiguity during re-factoring, we introduced decision tables. Given an input event, system context, and policy settings, the table in figure 3 looks up a set of actions. This combination of conditions is called a rule. In each cell, 0 is false; 1 is true. We coded the tables as XML files downloaded from the parts library server to the adapter framework on the desktop. The tables are interpreted at runtime.

USE CASE:		Rules			
UC.9	check in with dependency conflict	1	2	3	4
EVENT:					
E.1	draftsman checks in engine	1	1	1	1
CONDITION FLAGS (CONTEXT):					
C.1	matching file exists on server?	0	1	1	1
C.2	dependents checked out (locked)?	0	0	1	1
POLICY SETTINGS:					
P.1	notify dependents on lock state change	0	0	1	0
ACTIONS:					
A.1	initiate new file check in	1	0	0	0
A.2	cancel inactivity timer	0	1	1	1
A.3	unlock existing engine files on server	0	1	1	1
A.4	increment file version	1	1	1	1
A.5	notify manager that engine changed	0	0	1	0
A.6	prompt to merge car with new engine	0	0	0	1

Figure 3. Decision table for check-in use case

The decision table shown in figure 3 matches the protocol shown in figure 2. Note rules 3 and 4. When the draftsman attempts to check in the CAD model of an engine, and the car is checked out to someone else (C.2 is true), then policy P.1 determines whether action A.5 executes. This controls how users are notified to resolve check-in conflicts.

2.3. Results

Language, time zones, and cultural barriers led to the following project challenges:

- ***Ambiguity*** – Evolving requirements and misunderstandings complicated development.
- ***Quality issues*** – Our first solution, using hard-coded logic based on state variables that were passed back and forth was error prone.

We eliminated ambiguity by introducing decision tables and documents generated from code. This improved communication, experimentation, and software quality.

3. Design competitions

A design competition involves different teams (or different members of the same team) working independently to pursue different designs to a common problem. This works well on distributed teams since they can work in isolation. Once complete, the competing designs are reviewed and merged.

Despite good documentation and late-night conference calls, our distributed team was hindered by miscommunication, assumptions, and schemata (i.e., differences in how people approach problem solving).

After substantial design and prototype work, we applied Goldratt's Categories of Legitimate Reservation as an evaluation methodology. Briefly, this methodology seeks to identify system constraints by finding valid categories of criticism such as clarity (of an idea or system component), entity existence (missing component), cause-effect reversal (backwards data or control flow), etc. [8] We discovered that we had not fully explored the design space. Goldratt categorizes this problem as cause insufficiency (i.e., missing antecedent). Thus, our second collaboration strategy was to use competitive design to fully explore all options and make good design choices.

3.1. Problem description

There were several design options to integrating the CAD tool with our desktop software. Recall that our software provides a mechanical engineer with access an online parts library. The first option was to invoke our software synchronously from within the CAD tool, making the parts library browser appear as an embedded window within the CAD tool. The second option was to run our software as an applet within a separate browser process using remote procedure calls for inter-process communication with the CAD tool. We selected the second option in part because this loose coupling positioned our software to work with different CAD tools.

However, the second option turned out to have significant security issues. The applet had to do all of the things that a browser's security sandbox prohibits: it had to perform inter-process communication, file system I/O, and modify the browser process's environment variables.

3.2. Our approach

Our solution was adapted from a blog post on Java security. It enables an applet to bypass a browser's security sandbox [9]. The following code is deployed in a digitally signed jar file. Here is how it works:

On line 16, the applet creates a worker thread to do the tasks prohibited by the security sandbox.

```
1  public class CadApplet extends Applet {
2   private CadWorkerThread _consumer;
3   private Itm    _inputQueue;
4   private Itm    _outputQueue;
5
6   public class Itm { //InterThread Message
7    public static final int IDLE =0;
8    public static final int RESULT_READY =1;
9    public static final int getModel =2;
10   public static final int otherMethods =3;
11   public Integer type;
12   public String msg;
13  }
15  public void init() {
16   _consumer = new CadWorkerThread();
17   _consumer.start();
18  }
```

When a user visits the on-line parts library website, JavaScript (not shown) executes the applet's public methods. For example, to retrieve information (metadata) about a 3D model that is open in the CAD tool, the JavaScript invokes the public method applet.getModel() on line 22.

```
20  public String getModel(String params) {
21   Itm itm = new Itm();
22   itm.type = Itm.getModel;
23   itm.msg = params;
24   return producer(itm);
25  }
```

All public methods invoke producer(), which puts the user's request in a queue shared by worker thread (e.g., line 24). Then the main applet thread sleeps until the worker thread performs the requested task.

```
27  private String producer(Itm itm) {
28   String tmpString, result;
29   Integer tmpInteger;
30   synchronized (_inputQueue) {
31    _inputQueue = itm;
32   }
33   while (true) {
34    Thread.sleep(300);
35    synchronized (_outputQueue) {
36     tmpString = _outputQueue.msg;
37     tmpInteger = _outputQueue.type;
38     _outputQueue.msg = "";
39     _outputQueue.type = Itm.IDLE;
40    }
41    if (tmpInteger == Itm.RESULT_READY) {
```

```
42     result = tmpString;
43     break;
44   }}
45   return result;
46  }
```

The worker thread reads the request from the shared queue (lines 60, 61) and dispatches the request message to the correct handler (lines 64, 66).

```
48  class CadWorkerThread extends Thread {
49   public CadWorkerThread() {
50    _inputQueue = new Itm();
51    _outputQueue = new Itm();
52   }
53
54   public void run() {
55    String tmpString;
56    Integer tmpInteger;
57    while (true) {
58     Thread.sleep(300);
59     synchronized (_inputQueue) {
60      tmpString = _inputQueue.msg;
61      tmpInteger = _inputQueue.type;
62     }
63     if (_inputQueue.type != Itm.IDLE) {
64      switch (tmpInteger) {
65      case Itm.getModel:
66       tmpString = getModelImpl(tmpString);
67       break;
68      case Itm.otherMethods:
69      }
70      synchronized (_outputQueue) {
71      _outputQueue.type = Itm.RESULT_READY;
72      _outputQueue.msg = tmpString;
73  }}}}}
```

When the worker thread completes, the inter-thread Message.type is set to RESULT_READY and result is passed back to the main thread (lines 71, 72). The main thread returns the result to the calling JavaScript.

3.3. Results

Prior to our design competition, two barriers hindered remote collaboration:

- ***Mismatched skill sets*** – one offshore resource had excellent CAD skills, but did not have the programming experience required to solve problems with security and recursive model composition. Our onshore resources had these programming skills, but did not know the CAD programming interface. The root cause was an offshore hiring decision based on a telephone interview and a limited pool of qualified candidates. Instead of anticipated cost savings, we had unanticipated expenses and a minor change in product direction.
- ***Schemata differences*** – Based on prior personal experiences, different team members had different ways of thinking about solving problems and programming (e.g., composition and recursion, synchronous vs. asynchronous messaging, and single- vs. multi-threaded programming).

These barriers led to the following challenges:

- ***Rework*** – We implemented the applet twice to resolve security problems, a consequence of not fully exploring all possible solutions.
- ***Misunderstandings of design intent*** – Prior to the design competition, no one could see a viable solution to this problem. Only by giving both sites the freedom to independently pursue different approaches – and fail on their own – were we able to come together in the end and produce a working solution.

Several lessons came out of this experience. First, always use an evaluation framework such as "Goldratt's Categories of Legitimate Reservation" to review competing designs. Second, understand the true cost of offshore work. Barriers to collaboration may lead the team down a path with unanticipated costs. Finally, pair developers with different skills or ideas: each will compensate for the other.

4. Strategy maps

Towards the end of our project, we conducted a review to determine how our barriers to remote collaboration affected the project outcome. We created the strategy map in Figure 4, which traces aspects of software development across five perspectives of a business. For example, to make money (financial perspective), customers need to be happy (customer perspective). To satisfy customers, internal operations must be efficient (operations perspective). And that requires an agile team that is constantly learning (learning perspective). But, when the team is distributed, barriers (collaboration perspective) impact activities in the other perspectives. In our case, ambiguity adversely affected software quality. We were able to compensate by using extreme programming (XP) and agile practices to improve software quality.

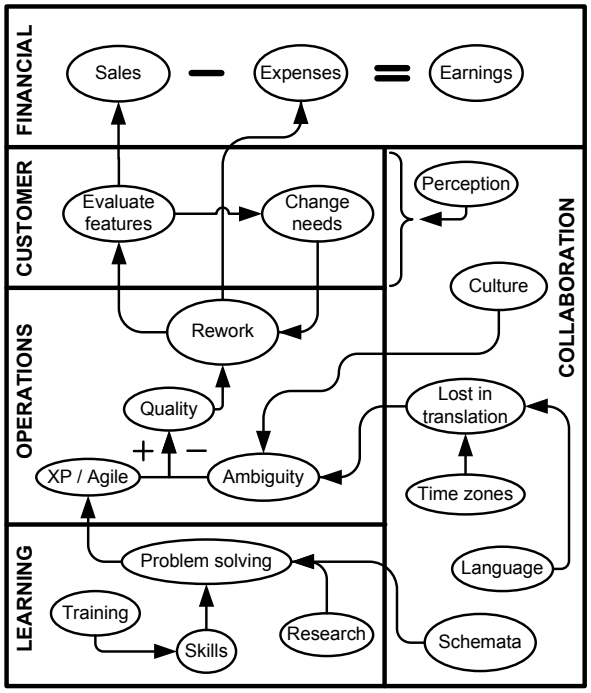

Figure 4. Strategy map showing collaboration

To produce the map, we combined Kaplan's Balanced Score Card strategy tool with Goldratt's Theory Of Constraints [3][8]. The author's contribution was to add the collaboration perspective.

4.1. Results

The strategy map with a collaboration perspective and balanced score card (not shown) revealed the true cost of offshore development. We used this to justify increased funding for the learning perspective, which is traditionally under-funded.

The strategy map also helped us build common understanding among all stakeholders. This was important because software engineers tend to think from the bottom up (i.e., implementation focus) while business users tend to think from the top down (i.e., feature focus in the context of business scenarios). Our development resources at offshore sites understood continuous integration, test-driven development, and data-driven architecture (internal operations perspective). And, our business users understood the value of a highly configurable system that permitted rapid prototyping of changing requirements (customer perspective). In hindsight, we should have created this strategy map at the beginning of the project.

5. Conclusion

Future work is needed in two areas. First, methods are needed to re-factor/simplify decision tables. Second, metrics are needed to quantify the collaboration perspective of strategy maps.

Before committing to a new offshore development project, understand the true cost of collaboration among geographically distributed team members. Create a strategy map for your specific project and build a business case for additional funding in the learning perspective. Then establish and execute your strategies to bridge the barriers to collaboration on your specific project. Finally, enjoy the challenges and rewards of collaborating on global teams. For further information, visit www.madsen.us.

References

[1] Carmel, E., Tjia, P. Off-shoring information technology. Cambridge, UK: Cambridge University Press, 2005, pp 12 and 14.

[2] http://en.wikipedia.org/wiki/Schema_(psychology)

[3] Kaplan, Robert and Norton, David, *The Strategy-Focused Organization*, Harvard Business School Press, Boston, 1998, pp 69-81.

[4] Gross, C., Reischl, U., and Abercrombie, P., The New Idea Factory. Battelle Press, Columbus, OH, USA, 2000, pp 3.

[5] Savitt, S, "Being and Becoming in Modern Physics" in the *Stanford Encyclopedia of Philosophy*, 2001 http://plato.stanford.edu/entries/spacetime-bebecome and http://plato.stanford.edu

[6] Beck, K., Extreme Programming Explained, Second Edition. Addison-Wesley, Upper Saddle River, New Jersey, USA, 2005, pp 30.

[7] Heun, W., "Systems Engineering of Complex Software Systems," 37th ASEE/IEEE Frontiers in Education Conference, October 10 – 13, 2007, Milwaukee, WI, pp 1.

[8] Dettmer, William, *Breaking the Constraints to World-Class Performance*, North River Press, 1998, pp 11 & 45.

[9] Applet Security discussion thread. We cannot find the original reference. So, we wish to give credit to all those who post ideas to discussion threads to advance the art and science of programming.